

---

# 2022 年度 情報理工学実験 I

提出日:2022/10/30

工学部 情報エレクトロニクス学科 情報理工学コース 3年

氏名: 豊田真吾

学籍番号: 02206909

---

## 実験テーマ ~データベースとWebサーバプログラミング~ (Part2)

---

## 基本課題 3-1

NoSQLはデータベース管理システムのうち、関係データベースであるRDBMS以外を指すものである。RDBMSは、SQLなどで知られているように、固定の列と行を持つテーブルにデータを保存する形式のデータベースの総称であり、1970年代ごろから利用されるようになった。これに対してNoSQLは、情報をテーブルで管理しない形式のデータベースであり、NoSQLという言葉はSQLインタフェースを持たない軽量な関係データベースのオープンソースソフトウェアの名前として1998年に最初に用いられた。NoSQLのデータベースには様々な形式のものがあるが、代表的なNoSQLにMongoDBなどがある。今回は**MongoDB**に注目してRDBMSとの比較や使用用途、実際の使用例を調査する。

### RDBMSと比較した長所

MongoDBはドキュメント型の構造的なデータをJSONのような形式で表現し、ドキュメントの集合を管理するドキュメント指向データベースである。RDBMSではデータベースをテーブル形式で表現するが、これにはスケールアウトが難しいという問題がある。

RDBMSではレコード数が増えれば増えるほど表がどんどん縦に伸びる。レコードが膨大になり過ぎたら、そのデータベースが置かれたサーバ自体の処理能力を増強するしかない。サーバを増強させることをスケールアップという。ただしスケールアップには限界があり、レコード数が膨大な数に増えていった場合には1台の高価なマシンでデータを管理するには多大なコストがかかってしまう。そこで複数台のマシンで情報を管理する“スケールアウト”を行って情報を管理することが必要となってくる。しかし、データベースを分割すると通信の失敗などで整合性が取れないなどの複雑な問題が無数にでてくるという欠点があり、データを正規化し厳格に管理するRDBMSにはこのスケールアウトが向いていないとされている。これと比較してMongoDBでは複数のノードによって異なるデータを保存する“シャーディング”という負荷分散のしくみを持っているため、複数のマシンでデータを扱いやすい。

また、MongoDBはデータベース側で行う処理をすべてメモリ上で実行することが可能である。したがって単純な読み込みや書き込みはメモリ容量の範囲内であれば高速に実行することができる。また、MongoDBをはじめとした他のNoSQLについても複雑なクエリ処理をせずに、膨大な量のデータをデータベースに書き込む速度や読み出す速度を重視しているという特徴がある。

データの保存形式にも大きな違いがあり、MongoDBではデータベースが構造を定義する情報を持たないスキーマレスという仕様で、JSONの形のまま不定形のデータを格納することができる。そのため、複雑なデータ構造であってもRDBMSのようにテーブルに分けて格納するのではなく、そのままデータベースに格納できることが可能である。

### RDBMSと比較した短所

---

RDBMSはデータの一貫性を保つために**ACID**というデータの不整合が置きない技術が実装されている。ACIDとは、

- **Atomicity**(原子性)
  - トランザクションが完了した時点において、トランザクションの中身がすべて実行完了したか、まったく実行されていないかのどちらかしか起こらないという性質
- **Consistency**(一貫性)
  - トランザクションの実行完了ごとにデータベースが一貫性を維持するという性質
- **Isolation**(独立性)
  - トランザクションの実行は、他のトランザクションから影響を受けず影響を与えないという性質
- **Duration**(永続性)
  - トランザクションの実行により行われたデータベースへの変更は永続するという性質

の4つの性質であり、RDBMSではこれらが意識されることによってデータの一貫性が保証されている。一方のMongoDBをはじめとするNoSQLは、水平分散させることができるように一貫性は諦めた仕様になっている。

また、MongoDBではSQLの言語を用いることができないため、テーブルの結合操作やGROUP By などのQuery関数を使うことができない。独自のデータベース言語を学ぶ必要がある。

## 想定している用途

MongoDbは近年のウェブサービスなどに多く採用されており、ソーシャルゲームなどにはよく採用されている。ゲームは短期間での開発が要求され、ゲームという特性上、ソフトウェアアップデート等のたびに仕様変更が頻繁に行われる。スキーマレスであるMongoDBはデータ構造の変更に柔軟に対応する事ができるため、このようなゲーム開発に向いている。

この他にも、MongoDBはスキーマレスがデータベースとなっているため、特にユーザーによって独自設定を定義するようなシステムでの利用に適している。そのため、アンケートシステムなどに向いている。MongoDBならユーザが生成した独自項目を検索することも可能である。

## 実際の使用例

MongoDBをaptやbrewなどでインストールを行い。シェル上でmongoコマンドからmongo dbを起動する。その後、use "データベース名"でデータベースを作成する。

```
> use my_db
```

createCollectionコマンドでデータベースの中にコレクションを作成する。実行結果がJSONで返却される。

```
> db.createCollection("users")
{ "ok" : 1 }
```

insertOneコマンドでコレクションの中にドキュメントを作成することができる。SQLとは異なり、列名やデータタイプを設定せずともデータ作成ができる。

```
> db.users.insertOne({ name: "Shingo Toyoda", age: 20})
{
  "acknowledged" : true,
  "insertedId" : ObjectId("5d941d8020153c73fb1b1acf")
}
```

作成したデータはfindコマンドで検索することができる。nameとageの他に\_idキーが自動で追加されていることもわかる。

```
> db.users.find()
{ "_id" : ObjectId("5d941d8020153c73fb1b1acf"), "name" : "Shingo Toyoda", "age" : 20 }
```

データを更新するときは、update コマンドを利用する。

```
> db.users.update({"name":"Shingo Toyoda"},{$set: {"age":3}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

## 基本課題 3-2

SQLインジェクションについて

### 攻撃方法

アプリケーションの脆弱性により本来の意図ではない不当な SQL 文が作成されてしまい、注入されることによって、データベースのデータを不正に操作される攻撃のことである。

データベースと連携して動作する多くのWebアプリケーションは SQL 文を使い、ユーザからの各種情報入力を元にしてサーバ側で処理し結果をユーザに返す。

Webアプリケーションに脆弱性が存在すると、アプリケーションが入力値を適切に処理せず、SQL文を誤った命令文として認識してしまい、データベースの操作が可能になることで、情報漏えいやデータの悪用につながる。

### 攻撃例

weak\_sqlite.rb は、標準入力からidとnew\_nameをカンマ区切りで受け取り、該当するidの名前をnew\_nameに更新するプログラムである。ここで、入力欄に「0 or 'A' = 'A',QWERTY」と入力することによって、実行されるSQL文は

```
UPDATE students SET name = \"QWERTY\" WHERE student_id = 0 or 'A' = 'A'
```

となり、全ユーザの名前をQWERTYに更新することができる。

この他にも、入力欄に「0 or 'A' = 'A'; DELETE FROM students,QWERTY」と入力することによって、実行されるSQL文は

```
UPDATE students SET name = \"QWERTY\" WHERE student_id = 0 or 1 = 1; DELETE FROM students
```

となり、テーブルを全削除することができる。

## 対策

プレースホルダを用いて実装を行うことでSQLインジェクションを回避することができる。具体的には、:から始まる代替文字列(プレースホルダ)を宣言し、後から実際の値をバインドさせれば new\_nameとidの部分をSQL文と解釈されなくなるため、SQLインジェクションを回避できる。

```
str = "UPDATE students SET name = ? WHERE student_id = ?"  
db.execute_batch(str,[new_name],[id])
```

## 基本課題 3-3

lab\_members.csvとlabs.csvが配置されているディレクトリで以下のSQL文を実行し、データベースを作成する。

```
~/D/H/B/デ/P/kadai3-3 >>> sqlite3 data.db  
SQLite version 3.37.0 2021-12-09 01:34:53  
Enter ".help" for usage hints.  
sqlite> CREATE TABLE labs(id integer PRIMARY KEY,lab_name text);  
sqlite> CREATE TABLE lab_members(id integer PRIMARY KEY,lab_id integer,member_name text, position text);  
sqlite> .separator ,  
sqlite> .import lab_members.csv lab_members  
sqlite> .import labs.csv labs
```

labs

id	lab_name
1	Mathematical Science
2	Advanced Electronics
3	Bioengineering
4	hci-lab
5	System Creation
6	High Performance Computing
7	Algorithm

lab\_members

id	lab_id	member_name	position
1	1	Brenton_Evans	D1
2	1	Spring_Brandi	D1
3	1	June_Killinger	D2
4	1	Ettie_Judd	D2
5	1	Adaline_Yazzie	M1
6	1	Leia_Driscoll	M1
7	1	Sal_Hadfield	M1
8	1	Randal_Spang	M1
9	1	Abdul_Dossey	M2
10	1	Ruth_Emmick	M2
11	2	Federico_Brumit	B3
12	2	Alicia_Weatherbee	B4
13	2	Zetta_Kreisel	B4
14	2	Alvera_Corle	D1
15	2	Antone_Thrash	D2
16	2	Mao_Winberg	D2
17	2	Lillia_Corzine	D3

実行した結果、上の画像のようなデータベースを作成することができた。SQL文を用いて lab\_members から、lab\_id が 1 の id とメンバー名を表示できることを確認した。

```
sqlite> SELECT id,member_name FROM lab_members WHERE lab_id = 1;
1,Brenton_Evans
2,Spring_Brandi
3,June_Killinger
4,Ettie_Judd
5,Adaline_Yazzie
6,Leia_Driscoll
7,Sal_Hadfield
8,Randal_Spang
9,Abdul_Dossey
10,Ruth_Emmick
```

## 基本課題 3-4

研究室名(lab\_name)を入力するとメンバー名(member\_name)一覧が表示できるプログラムを lab\_search.rb、メンバー名を入力すると、所属する研究室の他のメンバーの一覧が表示されるプログラムを member\_search.rb として実装を行った。ファイルは別途添付している。

lab\_search.rb を実行し、"hci-lab" と入力すると、hci-lab に所属している学生が次のように一覧表示される。

```
~/D/H/B/テ/P/kadai3-4 >>> ruby lab_search.rb
hci-lab
Ana_Forth
Greg_Woo
Dannette_Audia
Sherika_Sidle
Nichelle_Koser
Chia_Papageorge
Scot_Declue
Mitsue_Harmon
Kam_Crosby
Hallie_Heinz
Rose_Satcher
Milissa_Kuta
Lucile_Chung
Yanira_Lollar
Lavon_Delaney
Kristal_Don
Masahiro Kitagawa
Sho Mitarai
Tetsuo Ono
Daisuke Sakamoto
```

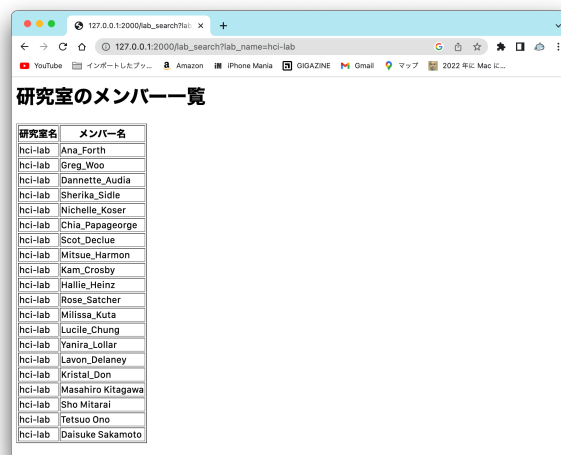
member\_search.rb を実行し、"Daisuke Sakamoto" と入力すると、Daisuke Sakamoto と同じ研究室に所属している他の学生が次のように一覧表示される。

```
~/D/H/B/テ/P/kadai3-4 >>> ruby member_search.rb
Daisuke Sakamoto
```

```
Ana_Forth
Greg_Woo
Dannette_Audia
Sherika_Sidle
Nichelle_Koser
Chia_Papageorge
Scot_Declue
Mitsue_Harmon
Kam_Crosby
Hallie_Heinz
Rose_Satcher
Milissa_Kuta
Lucile_Chung
Yanira_Lollar
Lavon_Delaney
Kristal_Don
Masahiro Kitagawa
Sho Mitarai
Tetsuo Ono
```

## 応用課題 3-1

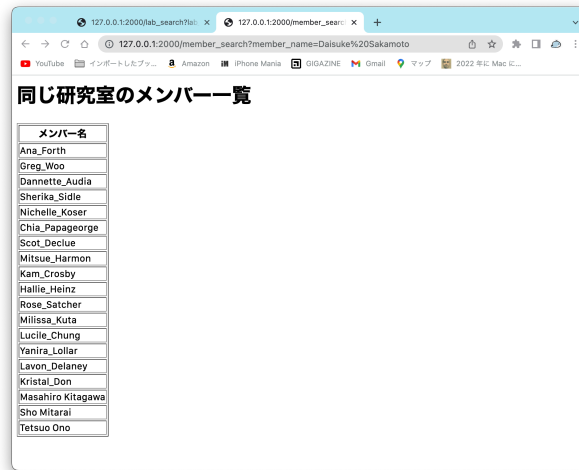
Web ブラウザから DB のデータを閲覧できるようなプログラムをlabSearch\_Server.rbとして実装した。/lab\_search にアクセスしてパラメータとして lab\_name を与えて研究室名をリクエストすると、その研究室のメンバー名一覧が取得される。lab\_nameが与えられなかったり、研究室が存在しなかった場合は"メンバーが見つかりませんでした。"と表示される。



研究室名	メンバー名
hci-lab	Ana_Forth
hci-lab	Greg_Woo
hci-lab	Dannette_Audia
hci-lab	Sherika_Sidle
hci-lab	Nichelle_Koser
hci-lab	Chia_Papageorge
hci-lab	Scot_Declue
hci-lab	Mitsue_Harmon
hci-lab	Kam_Crosby
hci-lab	Hallie_Heinz
hci-lab	Rose_Satcher
hci-lab	Milissa_Kuta
hci-lab	Lucile_Chung
hci-lab	Yanira_Lollar
hci-lab	Lavon_Delaney
hci-lab	Kristal_Don
hci-lab	Masahiro Kitagawa
hci-lab	Sho Mitarai
hci-lab	Tetsuo Ono
hci-lab	Daisuke Sakamoto

リクエスト例: [http://127.0.0.1:2000/  
lab\\_search?lab\\_name=hci-lab](http://127.0.0.1:2000/lab_search?lab_name=hci-lab)

また、/member\_search にアクセスしてパラメータとして member\_name を与えて研究室名をリクエストすると、その人が在籍している研究室の他のメンバー名一覧が取得される。member\_nameが与えられなかったり、メンバーが存在しなかった場合は"メンバーが見つかりませんでした。"と表示される。



リクエスト例: [http://127.0.0.1:2000/member\\_search?member\\_name=Daisuke%20Sakamoto](http://127.0.0.1:2000/member_search?member_name=Daisuke%20Sakamoto)

## 応用課題 3-2

掲示板サーバ `bulletinBoard_Server_withDB.rb` を作成した。ただし、応用課題2-2にて投稿の削除を実装しているため、書き込み「`write`」、書き込み一覧取得「`index`」、書き込み取得「`read`」に加えて書き込み削除「`delete`」機能も引き続き実装している。

まずはじめにメッセージを管理するDBを作成する。テーブルは1つで、メッセージのID(`id`), ユーザ(`user`), 本文(`msg`), の3つのデータを保存する。

```
~/D/H/B/デ/P/ouyou3-2 >>> sqlite3 bulletinBoard.db
SQLite version 3.37.0 2021-12-09 01:34:53
Enter ".help" for usage hints.
sqlite> CREATE TABLE messages(id integer PRIMARY KEY,user text,msg text);
sqlite>
```

### 書き込み

パラメータとして`user`と`msg`を与えて/`write`にアクセスすることで書き込みを行う。`user`と`msg`どちらかのパラメータが与えられていない場合や空白である場合は書き込みを行わない。

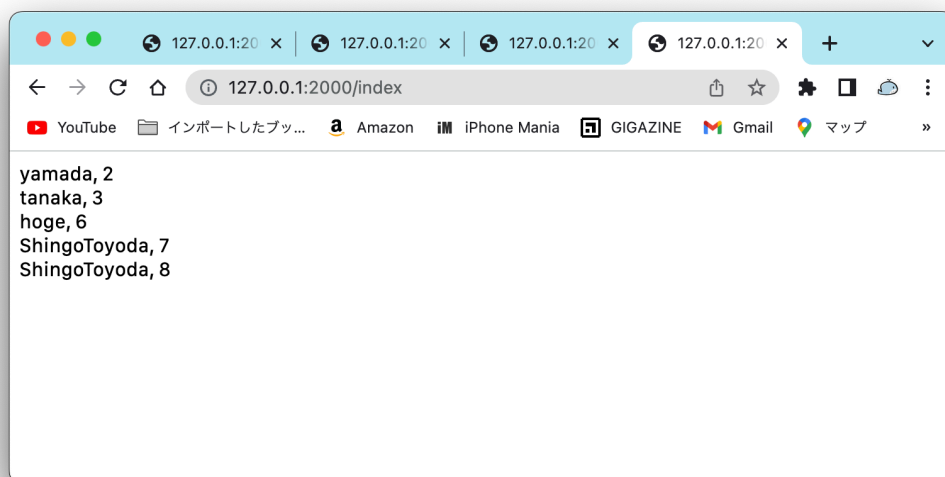




アクセス例: <http://127.0.0.1:2000/write?msg=%E3%81%93%E3%82%93%E3%81%AB%E3%81%A1%E3%81%AF&user=ShingoToyoda>

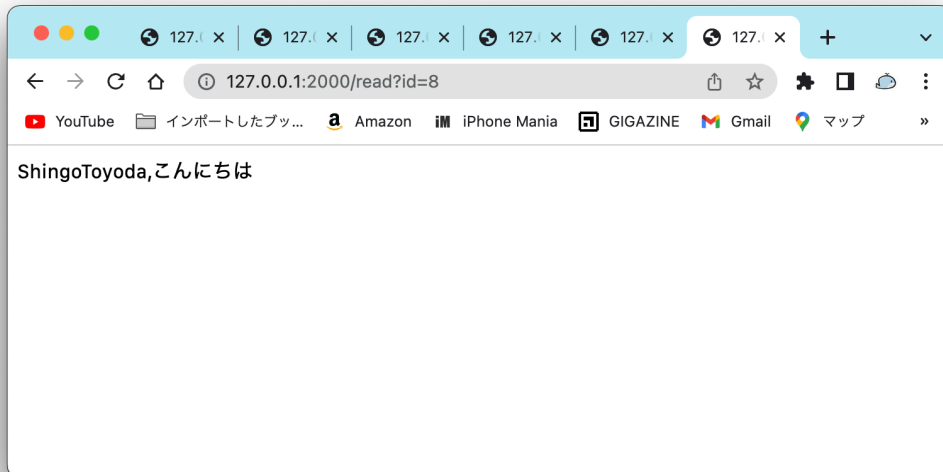
## 書き込み一覧取得

/indexにアクセスすることでメッセージのユーザ名とIDが一覧表示される。



## 書き込み取得

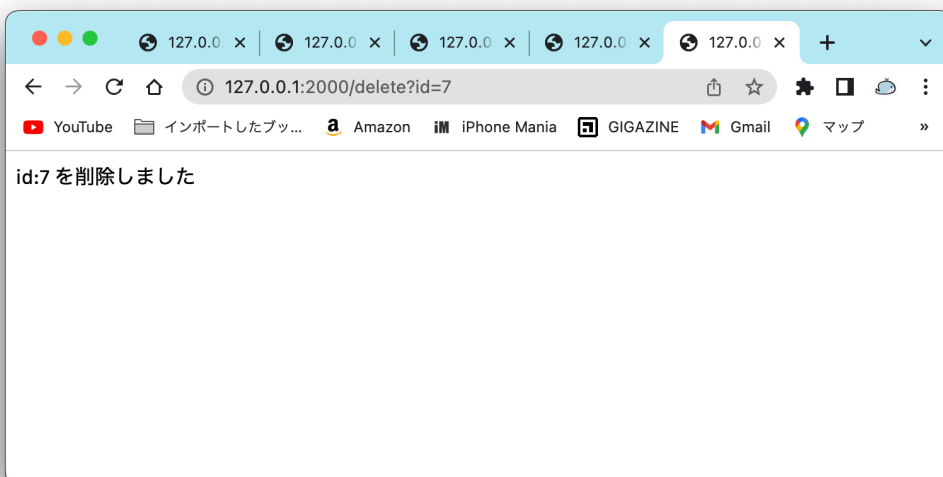
パラメータidを与えて/readにアクセスすることによってユーザ名と本文が表示される。メッセージが存在しない場合には"メッセージが見つかりませんでした"と表示される。



アクセス例: <http://127.0.0.1:2000/read?id=7>

## 書き込み削除

パラメータidを与えて/deleteにアクセスすることによってメッセージが削除される。メッセージが存在しない場合には"メッセージが見つかりませんでした"と表示される。



アクセス例: <http://127.0.0.1:2000/delete?id=7>

---

## 応用課題 3-3

ユーザ機能を追加したログイン機能を作成した。ユーザ情報のデータベースを作成し、次のように(ユーザ名,パスワード) = (A,hoge) , (B,hirakegoma) とするテーブルを作成した。

```
~/D/H/B/デ/P/ouyou3-2 >>> sqlite3 password.db
SQLite version 3.37.0 2021-12-09 01:34:53
Enter ".help" for usage hints.
sqlite> CREATE TABLE passwds(id integer PRIMARY KEY,user text,password text);
sqlite> INSERT INTO passwds(user,password) VALUES("A","hoge");
sqlite> INSERT INTO passwds(user,password) VALUES("B","hirakegoma");
```

ログイン画面にて正しいパスワードとユーザ名の組を入力すると、OKと表示され、誤ったパスワードとユーザ名を入力するとNGと表示される。

## 参考文献

- NoSQLとはなにか - <https://www.mongodb.com/ja-jp/nosql-explained>
- MongoDBを用いたモバイルゲーム開発について - <https://ameblo.jp/principia-ca/entry-10983188853.html>
- MongoDB超入門 - <https://qiita.com/saba1024/items/f2ad56f2a3ba7aaf8521>